

# A CUBIC TRANSITION CURVE TO BRIDGE A DISCONTINUITY

MARC A. MURISON

Astronomical Applications Department, U.S. Naval Observatory, Washington, DC 20392  
murison@aa.usno.navy.mil

March 9, 1999

## ABSTRACT

I show for reference a quick and painless way to smoothly bridge a discontinuity in a function. This is useful, for example, in the numerical integration of force functions containing discontinuities.

*Key words:* curves — ODE — numerical integration

### 1. INTRODUCTION

Sometimes it is necessary to integrate a set of ordinary differential equations (ODEs) that has a discontinuity, for example stochastic forces randomly sampled from a distribution. Generally, ODE integrators (e.g., Bulirsch-Stoer or even the normally robust Runge-Kutta) are very unhappy when they encounter discontinuities in a force function. One way around this problem is to introduce a finite transition time and then a continuous function that bridges the transition across the discontinuity. The simplest smoothing function is a third-order polynomial that matches the values and the derivatives at the two endpoints.

### 2. AN ENDPOINT-MATCHING CUBIC POLYNOMIAL

Suppose we have a function  $f(x)$  which is discontinuous at  $x=x_1$ . As a practical matter, we must insert a transition interval  $\Delta x$  over which to join the two discontinuous values, say  $\Delta x=x_2-x_1$ . Let our smoothing function be a cubic polynomial,

$$p(x) = ax^3 + bx^2 + cx + d \quad (1)$$

Then  $p(x)$  and  $dp(x)/dx$  must match  $f(x)$  and  $df(x)/dx$  at the discontinuity transition endpoints  $x_1$  and  $x_2$ . In other words, we require

$$\left. \begin{aligned} p(x_1) &= f(x_1) \\ p(x_2) &= f(x_2) \\ \left. \frac{dp}{dx} \right|_{x_1} &= \left. \frac{df}{dx} \right|_{x_1} \\ \left. \frac{dp}{dx} \right|_{x_2} &= \left. \frac{df}{dx} \right|_{x_2} \end{aligned} \right\} (2)$$

Substituting eq. (1) into eqs. (2), solving for the coefficients, and putting those back into eq. (1), we find the solution

$$\begin{aligned} f(x) = & \left[ (x_2 - x_1) \left( \frac{df}{dx_1} + \frac{df}{dx_2} \right) - 2(f(x_2) - f(x_1)) \right] \frac{x^3}{(x_2 - x_1)^3} \\ & + \left[ 3 \frac{x_1 + x_2}{x_2 - x_1} (f(x_2) - f(x_1)) - (x_1 + x_2) \left( \frac{df}{dx_1} + \frac{df}{dx_2} \right) \right. \\ & \quad \left. - \left( x_2 \frac{df}{dx_1} + x_1 \frac{df}{dx_2} \right) \right] \frac{x^2}{(x_2 - x_1)^2} \\ & + \left[ \frac{x_1 + x_2}{x_2 - x_1} \left( x_2 \frac{df}{dx_1} + x_1 \frac{df}{dx_2} \right) + \frac{x_1 x_2}{x_2 - x_1} \left( \frac{df}{dx_1} + \frac{df}{dx_2} \right) \right. \\ & \quad \left. - 6 \frac{x_1 x_2}{(x_2 - x_1)^2} (f(x_2) - f(x_1)) \right] \frac{x}{x_2 - x_1} \\ & - \frac{1}{(x_2 - x_1)^2} \left[ \frac{x_1^3 f(x_2) - x_2^3 f(x_1)}{x_2 - x_1} \right. \\ & \quad \left. + x_1 x_2 \left( x_2 \frac{df}{dx_1} + x_1 \frac{df}{dx_2} - 3 \frac{x_1 f(x_2) - x_2 f(x_1)}{x_2 - x_1} \right) \right] \end{aligned} \quad (3)$$

Figure 1 is a plot illustrating this cubic transition function from  $x_1=0$  to  $x_2=1$ .

### 3. A C++ IMPLEMENTATION

The following C++ code evaluates the cubic solution, eq. (3), at the point  $x$ , given the two endpoints  $x_1$  and  $x_2$ , the two function values  $f_1$  and  $f_2$  at the endpoints, and the derivatives of the function  $df_1$  and  $df_2$  at the endpoints. It has been optimized by Maple, so it is fast but not particularly human-readable.

```
inline double cubic( double x,
    double x1, double x2, double f1,
    double f2, double df1, double df2 )
{
    double t7 = x2*x2;
    double t16 = t7*x1;
    double t8 = x1*x1;
    double t15 = x2*t8;
    double t13 = x*x;
    double t14 = t13*x;
    double t6 = 2.0*x1+x2;
    double t5 = -x2+x1;
    double t3 = x1+2.0*x2;
```

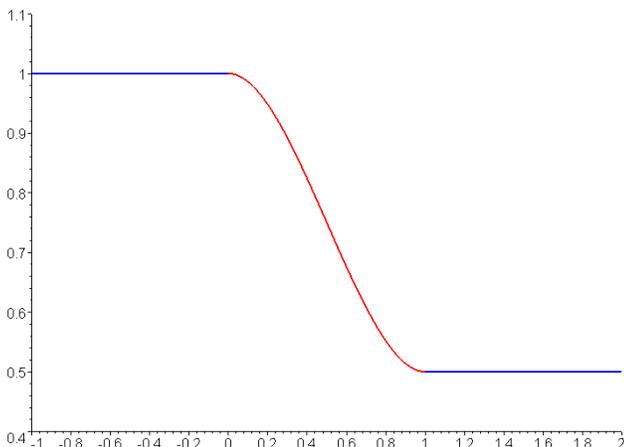


Figure 1

```

double t1;
t1 = ( ( (t8*x1-3.0*t15)*f2 +
        (-x2*t7+3.0*t16)*f1 +
        (2.0*t14 + 6.0*x1*x2*x -
         3.0*(x1+x2)*t13)*(f2-f1)
        )/t5 +
      (df1+df2)*t14 +
      (-t3*df1-t6*df2)*t13 +
      (t6*x2*df1+t3*x1*df2)*x -
      df1*t16 - df2*t15
      )/(t5*t5);
return(t1);
}

```

The function below, adapted from a working project that implements this transition curve scheme in a numerical integration of solar wind forces, illustrates how one might implement a finite transition time at discontinuities. The problem is not quite a straightforward one because most numerical ODE integrators calculate the force function at several substeps during a timestep, and the sequence of substeps may or may not span the beginning or end of a discontinuity and all or part of its transition curve. In this particular example, the force is constant between the discontinuities and outside the transition curves, similar to the example shown in Figure 1.

At some integration time  $t$ , the variable  $t\_changed$  is the time of the *previous* discontinuity,  $transition\_time$  is the (possibly artificial) amount of time introduced to bridge the

gap between discontinuity values,  $dt\_change$  is the amount of time from  $t\_changed$  to the next discontinuity,  $F$  is the current value of the force,  $F\_new$  and  $F\_old$  are then the new and old values, and  $deriv()$  is the function that calculates the new discontinuous force value. All of these variables are encapsulated C++ class data. In a C program, they would have at least file scope.

```

void StateVector::ForceFunction( double t )
{
    double t1 = t_changed;
    double t2 = t1 + transition_time;
    double t3 = t1 + dt_change;
    if( t < t1 ) {
        F = F_old;
    } else if( t > t1 && t < t2 ) {
        F = cubic(t,t1,t2,F_old,F_new,0,0);
    } else if( t > t3 ) {
        F = deriv(t);
        F_old = F_new;
        F_new = F;
        t_changed = t3;
    } else {
        F = F_new;
    }
}

```